9-2020

# Effective Activation Functions for Homomorphic Evaluation of Deep Neural Networks

Srinath Obla
sso8260@rit.edu

# Effective Activation Functions for Homomorphic Evaluation of Deep Neural Networks

**by**

**Srinath Obla**

**THESIS**

Presented to the Faculty of the Department of Computer Science

Golisano College of Computer and Information Sciences

Rochester Institute of Technology

in Partial Fulfillment

of the Requirements

for the Degree of

**Master of Science in Computer Science**

**Rochester Institute of Technology**

September 2020

# Acknowledgments

Looking back, working on this research has been an incredible journey and has taught me just as much as all the other courses combined, if not more. There are many I would like to thank without whom, I would not have been able to present these results.

Firstly, I owe my deepest gratitude to my Advisor, Dr. Peizhao Hu for his immense support, direction and patience. His guidance enabled me to push through roadblocks, publish my research and culminate this thesis. I would also like to offer my thanks to Dr. Ifeoma Nwogu for her suggestions and support in analyzing my experiment results. Many thanks to Dr. Stanislaw Radziszowski and Dr. Yu Kong for their help, feedback and for being on my committee.

I received generous support from my colleagues at RIT; Discussions with Viktoria Koscinski and Xinghan Gong have been very insightful and their support in collecting data and the necessary mathematical background has been invaluable. I have also had the constant aid and encouragement from Cindy Wolfer as my Graduate Advisor without which, this thesis would not have concluded. Special thanks to Dr. Hans-Peter Bischof for his support.

My deepest appreciation to those close to me who cheered on regardless of the ups and downs; Especially, Pratishta Rao for her steadfast belief in me

and her constant support for every late night experiment run, and Abishai Dmello, for his vital comments and suggestions.

Finally, I owe my deepest gratitude to my parents, Srikumar and Rajashree Obla, and my sister, Harini Obla. It is only because of their support, sacrifice and hard work that I had the opportunity to pursue and complete my thesis.

**Abstract**

# Effective Activation Functions for Homomorphic Evaluation of Deep Neural Networks

Srinath Obla, M.S.
Rochester Institute of Technology, 2020

Supervisor: Dr. Peizhao Hu

CryptoNets and subsequent work have demonstrated the capability of homomorphic encryption (HE) in the applications of private artificial intelligence (AI). While convolutional neural networks (CNNs) are primarily composed of linear functions which can be homomorphically evaluated, layers such as the activation layer are non-linear and cannot be homomorphically evaluated. One of the most commonly used alternatives is approximating these non-linear functions using low-degree polynomials. However, it is difficult to generate efficient approximations and often, dataset specific improvements are required. This thesis presents a systematic method to construct HE-friendly activation functions for CNNs. We first determine the key properties in a good activation function that contribute to performance by analyzing commonly used functions such as Rectified Linear Units (ReLU) and Sigmoid. We

then analyse the inputs to the activation layer and search for an optimal range of approximation for the polynomial activation. Based on our findings, we propose a novel weighted polynomial approximation method tailored to this input distribution. Finally, we demonstrate effectiveness and robustness of our method using three datasets; MNIST, FMNIST, CIFAR-10.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Deep neural networks have made significant contributions to solving complex tasks, especially in computer vision. In some applications, these networks require a large volume of private data; for example, lung cancer [70] and diabetic retinopathy detection [62]. However, regulations such as HIPAA [1] and GDPR [32] impose stringent restrictions on the use of private data by third-party service providers.

Homomorphic encryption (HE) is a privacy-preserving cryptographic scheme that supports arithmetic operations, such as addition and multiplication, on encrypted data without decrypting it first. Given two messages $m$ and $m'$ and the operations homomorphic addition $\oplus$ and homomorphic multiplication $\otimes$, we have $Enc(m) \oplus Enc(m')$ which decrypts to $m + m'$, and $Enc(m) \otimes Enc(m')$ which decrypts to $m \times m'$. A deep learning pipeline reduced to only these operations can perform a task on encrypted data without decrypting it. For simplicity, we will use normal arithmetic operators to represent homomorphic operations in the remainder of this document.

Most of the components within a CNN comprise of linear functions which can be homomorphically evaluated using homomorphic addition and

multiplication. However, CNNs also consist of layers that are dependent on non-polynomial functions such as the activation layer. Such layers cannot be reduced to the permitted operations listed above and hence cannot be evaluated without a HE-friendly replacement.

## 1.1 HE-friendly alternatives for non-polynomial functions

In the pursuit of successfully evaluating CNNs on homomorphically encrypted data, developing effective and HE-friendly support for non-polynomial functions has been an active topic of research in recent years [13, 16, 17, 28, 38]. So far, the methods used to generate HE-friendly alternatives can be grouped into three categories:

- using a power function [28]
- discretely sampling the function to be replaced and using a look-up table to access these values [17],
- using low-degree polynomial approximations of the activation [13, 16, 38].

When CryptoNets [28] was first introduced, it used the power function with the degree 2, *i.e.* the square function $f(x) = x^2$, as an HE-friendly replacement for commonly used activation functions. Unfortunately, due to the exponential growth of this function the network training process is unstable and affects the efficiency of the CNN. As a consequence, CryptoNets cannot

sustain more than one HE-friendly activation layer and therfore suffers a significant impact on accuracy (98.95% compared to 99.77% in state-of-the-art performance) on the MNIST handwritten digit dataset [47]).

An alternative approach [17] samples the activation function as discrete values and stores these values in a look-up table for use during inference. During homomorphic evaluation, the program performs a table lookup obliviously. However, sampling at discrete intervals reduces the floating-point precision of the outputs which directly affected the performance of the neural network. Increasing the rate of sampling would result in larger lookup tables which would lead to longer processing times.

A more common approach is to generate a low-degree polynomial approximation of a traditional activation function and use it to evaluate on encrypted data. This approach is similar to using a power function but without the immediate exponential growth. Chabanne *et al.* [13] investigated the effectiveness of using Taylor-series polynomials to approximate the ReLU activation function. Together with other techniques such as batch normalization (BN) [40] which puts acceptable bounds on the inputs to the activation layer, these polynomials allow them to train deeper CNNs effectively. As a result, their networks were able to achieve a 99.30% classification accuracy on the MNIST dataset. CryptoDL [38] examined approximation methods using standard and modified Chebyshev polynomials in addition to the Taylor series approach. The study evaluted the effectiveness of approximating different activation functions such as ReLU, Sigmoid, Tanh on the MNIST and

CIFAR-10 [42] datasets.

From the existing works, we note that the polynomial approximation approach is more robust than the table lookup approach and yields higher accuracy than simply using a power function. But these works also show that identifying the best performing polynomial is a difficult task and often has to be customized for a dataset.

## 1.2 Contributions

This thesis focuses on developing a systematic method to construct effective HE-friendly activation functions for CNNs using polynomial approximations. While pursuing this goal, we have made the following contributions:

- We study the characteristics of traditionally used activation functions in CNNs. For example, we discuss how activation function with bounded derivatives have an effect on the training process and is crucial for achieving high accuracies in CNNs. We also analyse the effect of different datasets on the distribution of inputs to the activation layer and share this knowledge for future work.

- We leverage the above findings to propose a multi-polynomial system for larger and complex datasets such as CIFAR-10. Our experiments show that this approach yields an improvement over using a single polynomial approximation for the entire network.

- We also propose a weighted approximation technique for finding effective HE-friendly activation functions. Our experimental results obtained from the three datasets (MNIST, FMNIST and CIFAR-10) show that our proposed method is able to generate HE-friendly activation functions that yield higher or the same accuracy as other dataset specific polynomial approximations in the state-of-the-art works.

# Chapter 2

# Background

## 2.1 Background Concepts

In this section, we provide brief descriptions of the background and preliminaries of convolutional neural networks, homomorphic encryption, and methods to approximate functions using polynomials.

### 2.1.1 Convolutional Neural Networks

A CNN is a type of deep neural network that is used primarily to analyze image data and is composed of a stack of layers, as shown in Fig. 2.1. CNNs are first trained on a dataset and then can be use to perform inference on related data. CNNs transform the image data from the input layer, through layers of computations, into the scores of each label in the output layer. Each layer consists of weights $\omega$ that are adjusted during the network training phase using a technique called backpropagation [66]. In backpropagation, the weights corresponding to the transformations are adjusted to optimize a loss function based on the error between the prediction made by the network and the ground truth. More formally, we model a CNN as a sequence of transformations, such that at a layer $\ell-1$ some functions $f$ apply computations to the $g$ input neurons $x_1^{\ell-1}, x_2^{\ell-1}, ..., x_g^{\ell-1}$, yielding the value of $h$ output neurons $x_j^{\ell} = f(x_i^{\ell-1}); i \in$

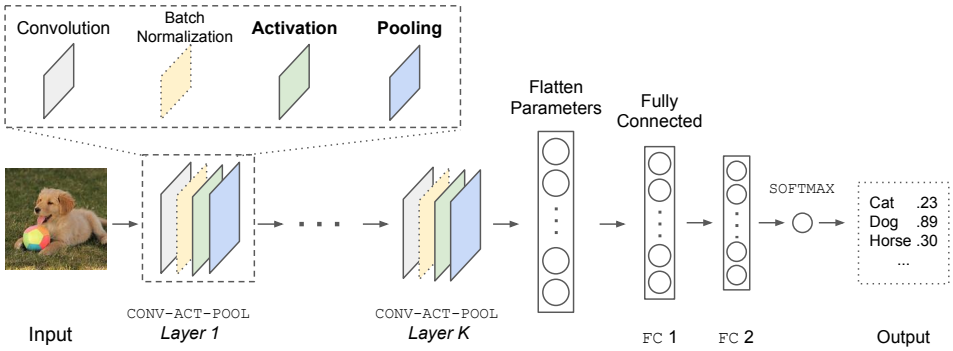



Figure 2.1: Typical architecture of a CNN

$\{1, ..., g\}; j \in \{1, ..., h\}$ at the layer $\ell$. Depending on the type of a the layer, the function $f$ consists of either a linear or non-linear transformation, as we will discuss below.

#### 2.1.1.1 Convolutional layer (CONV)

A CONV layer extracts hierarchical features from multiple representations of the input image by applying multiple filters on it as illustrated in Fig. 2.2. During the training process, the filters weights are adjusted in the optimization process to reduce the error function. In every filter, each weight $\omega^{\ell}_{i'j'k'}$ and bias $\beta^{\ell}_{i'j'k'}$, where $i', j' \in s; k' \in d$ at layer $\ell$, is learned from the training dataset. For the value of each neuron at layer $\ell$, we compute $x^{\ell}_{i'j'k'} = \beta^{\ell}_{k'} + \sum_{i=1}^{s}\sum_{j=1}^{s}\sum_{k=1}^{d} \omega^{\ell,k'}_{ijk} x^{\ell-1}_{i'+i-\lfloor s/2 \rfloor, j'+j-\lfloor s/2 \rfloor, k}$ using $k'$ filter and some neurons of the previous layer at $\ell - 1$. Essentially, a convolution operation consists of many dot-product over matrices of weights $\omega^{\ell}_{i'j'k'}$ of a filter and the elements from the region this filter has been applied to the previous layer, $x^{\ell-1}_{i'+i-\lfloor s/2 \rfloor, j'+j-\lfloor s/2 \rfloor, k}$, as illustrated in Fig. 2.2. When done, we slide the filter

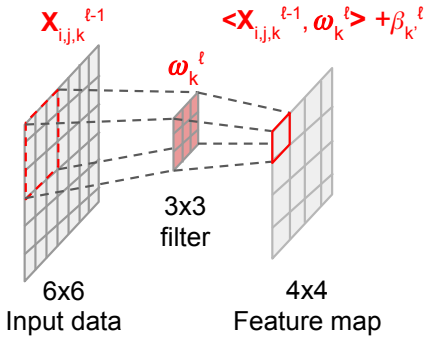



Figure 2.2: Convolutional filtering.

to the right to calculate the value for the next position in the feature map. This layer consists only linear transformations, hence it can be supported homomorphically.

#### 2.1.1.2 Batch normalization layer (BAT)

This operation was introduced by Ioffe and Szegedy [40] to accelerate the training of deep neural networks. The input to the batch normalization layer is transformed to have zero mean and unit variance using the statistics observed in a batch. Normalizing the data in this manner makes the neural network optimization smoother [67] and results in stable and predictive training. During inference, the network uses the moving averages learned during training to normalize the data. Algorithm 1 illustrates the calculations taken place in the batch normalization. It is important to note that the scaling and offset weights, $\gamma$ and $\beta$ are parameters obtained from training. This allows the network to not perform any normalization when it is desirable. While the BAT layer is optional, we depend on this layer to restrict the input distribution to

the activation layer.

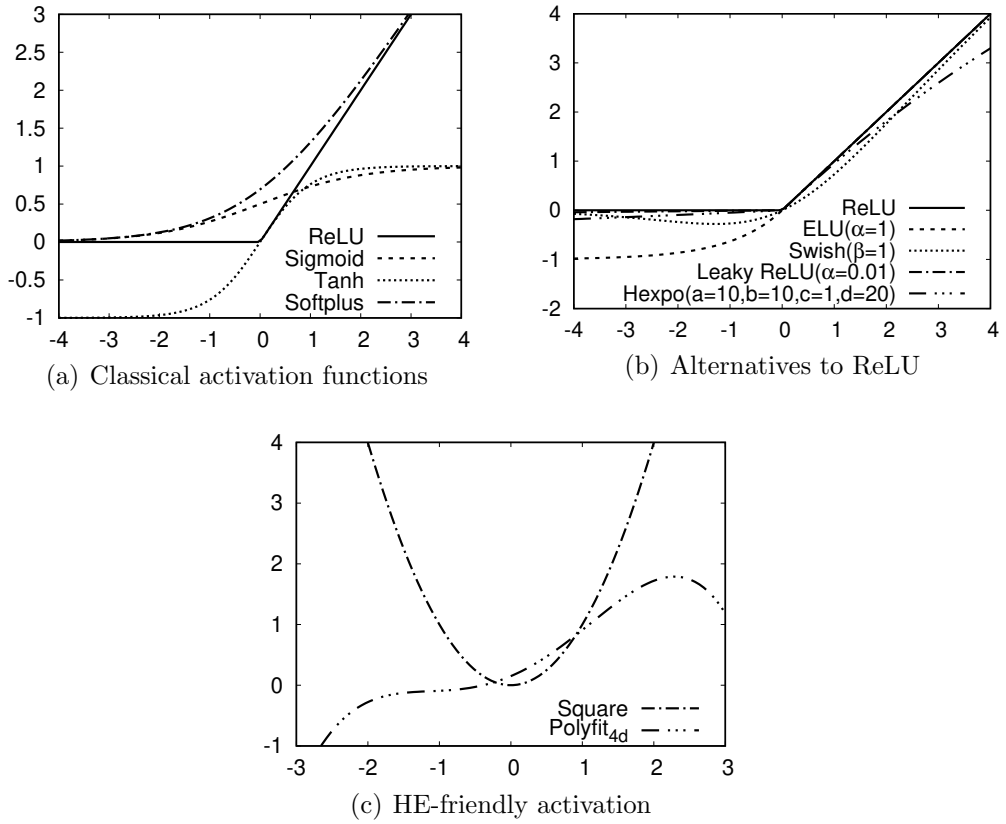
**Input:** $X = \{x_1, ..., x_m\}$
**Output:** $Y = \{y_i\}$
1: Calculate mean: $\mu \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
2: Calculate variance: $\sigma^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$
3: Normalize: $\hat{x}_i \leftarrow \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$
4: Scale and shift: $y_i \leftarrow \gamma\hat{x}_i + \beta$

**Algorithm 1:** Batch normalization over a mini batch $X$

#### 2.1.1.3 Activation layer (ACT)

The activation layers are an essential component in CNNs as their non-linear property permits the model to capture complex patterns from the input data. We compute the value of each neuron at the current layer $\ell$ as $x_{i'}^{\ell} = f(x_i^{\ell-1}); i \in g; i' \in h$, where $g$ are the neurons from the previous layer $\ell - 1$ and $h$ are the neurons from $\ell$. Depending on the application, there are various activation functions that can be used. In this research, we take a general stance and focus on the following classical activation functions as they often are augmented to other variants:

- Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$
- Logistic function (Sigmoid): $f(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent (Tanh): $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$
- Softplus: $f(x) = \log(1 + e^x)$




(a) Classical activation functions

(b) Alternatives to ReLU

(c) HE-friendly activation

Figure 2.3: Activation functions and alternatives.

Fig. 2.3(a) shows these activation functions for comparison. Among these activation functions, Sigmoid and similar functions such as Tanh often suffer from the vanishing gradient problem, which can potentially slow down the update of weights and biases, especially for deep networks. As observed from Fig. 2.3(a), Sigmoid and Tanh become saturated away from the origin; hence they get relatively small gradients. Activation functions like ReLU and it's variants solve this problem using rectification and linear responses as shown in Fig. 2.3(b).

There are multiple variants of classical activation functions like ReLU and Tanh optimized for specific applications with the addition of trainable parameters. However, as the focus is to support HE-CNNs from a general perspective, we choose to focus on the classical activations for our analysis and experiments. Fig. 2.3(c) shows two examples of these activation functions that are HE-friendly. Among them, CryptoNets [28] uses a square function but achieves low accuracy. Chabanne *et al.* [13] uses a degree 4 polynomial that approximates ReLU and achieves better accuracy than CryptoNets.

#### 2.1.1.4 Pooling layer (`POOL`)

Pooling layers are used in CNNs to reduce the amount of data between layers. The use of this operation results in the reduction of the number of weights, which helps the optimization process. Applying pooling layer allows a larger number of filters to be used in deeper convolution layers which help in extracting complex features. A pooling layer is commonly used after an activation layer. Together with the `CONV` and `ACT` layers, they form a structure which is repeated through the network; i.e., `CONV-ACT-POOL`. Generally, pooling layers use a window based approach to reduce local information. Depending on the type of pooling layer, a single value is generated for a region.

Two commonly used pooling functions are max-pool and average-pool. The average-pool function $f(X) = \frac{\sum_{i=1}^{n} x_i}{n}; x_i \in X$ can be computed easily, but the max-pool function $f(X) = \max(x_1, x_2, ..., x_n); x_i \in X$ cannot be reduced using the available operations and therefore would require developing a

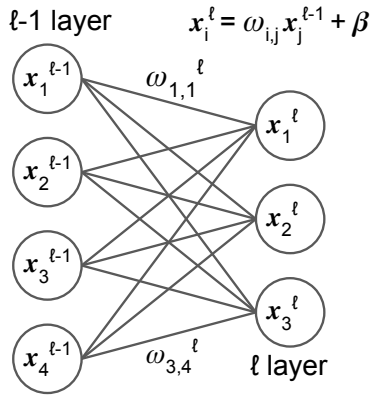



Figure 2.4: Fully connected layer.

replacement. Unlike activation functions, max-pool is multivariate and hence cannot be approximated directly using a single polynomial.

#### 2.1.1.5 Fully Connected layer (FC)

Every neuron $x_i^{\ell\prime}; i' \in h$ in a fully connected layer $\ell$ is connected to every other neuron $x_i^{\ell-1}; i \in g$ in the previous layer. Each connection has a trainable weight $\omega_{i'j'}^{\ell}$ and bias $\beta_i^{\ell\prime}$ associated with $x_i^{\ell\prime}$. We calculate the inner product yielding $x_{i'}^{\ell} = \beta_{i'}^{\ell} + \sum_i \omega_{i'}^{\ell} x_i^{\ell-1}$. Figure 2.4 depicts a common structure of neurons, weights and connections in a fully connected network.

#### 2.1.1.6 Softmax layer (SOFTMAX)

The softmax operation is commonly used in deep neural networks to transform the final outputs into a set of probabilities. Usually used in classification problems with more than 2 classes, this linear transformation converts all values from the previous layer to the range $(0, 1)$ but they sum to 1. Each of

these values can be interpreted as the probability score of the input belonging to the corresponding output class. Consequently, the class with the highest probability is the class prediction made by the neural network. More formally, the `SOFTMAX` layer can be represented as

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{k} e^{x_i}} \text{ where } j = \{1, ..., k\} \tag{2.1}$$

### 2.1.2 Homomorphic Encryption

HE is a class of encryption schemes that support computations such as addition and multiplication on encrypted data. Existing HE schemes can be divided into three main types based on the homomorphic operations supported by the evaluation function. In Partial HE (PHE) schemes, the evaluation function supports either addition (i.e. additive homomorphism), such as Goldwasser-Micali [31] and Paillier cryptosystem [60], or multiplication (i.e. multiplicative homomorphism), such as ElGamal cryptosystem [21], but not both. In contrast, Fully HE (FHE) allows arbitrary number of additions and multiplications. Somewhat HE (SWHE) schemes support both addition and multiplication on the ciphertexts. Yet, the number of multiplications allowed is limited due to the inherited construction of the scheme where ciphertexts contain noise that exponentially scales with multiplications. In general, HE scheme is a tuple of $\mathsf{PPT}$ algorithms $\mathsf{HE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$. We define each algorithm as follow.

- $\mathsf{HE.KeyGen}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$: Given a security parameter $\lambda$ determining

the security level, the key generation algorithm outputs a public key $\mathsf{pk}$, a private key $\mathsf{sk}$.

- $\mathsf{HE.Enc}(\mathsf{pk}, m) \rightarrow c$: Given a public key $\mathsf{pk}$ and a message $m$, the encryption algorithm outputs a ciphertext $c$.

- $\mathsf{HE.Eval}(\mathsf{pk}, f, c, c') \rightarrow c_{\mathsf{eval}}$: Given a public key $\mathsf{pk}$, two ciphertexts $c, c'$, and the homomorphic function $f$, the evaluation algorithm outputs the evaluated ciphertext $c_{\mathsf{eval}} = f(c, c')$.

- $\mathsf{HE.Dec}(\mathsf{sk}, c) \rightarrow m$: Given a ciphertext $c$ encrypted under $\mathsf{pk}$ and the corresponding secret key $\mathsf{sk}$, the decryption algorithm outputs the message $m$.

Note, the $\mathsf{HE.Eval}$ algorithm homomorphically performs a defined function $f$ on the ciphertexts. This function is constructed using $\mathsf{HE.EvalAdd}$ and $\mathsf{HE.EvalMult}$ which are homomorphic addition and multiplication respectively. Many HE schemes instantiate these common algorithms for integer-based computations, such as the BGV [10] and BFV [9, 23] schemes. Another HE scheme, the CKKS scheme [15], computes on fixed-point arithmetic, which is most appropriate for scientific research that often deals with floating-point data. Most well-known HE libraries (e.g., Palisade [3], Microsoft SEAL [69], and HElib [2]) have support for CKKS. Further reading on the construction of HE schemes can be found in the following survey [4, 6, 25, 55, 74].

Since we cannot homomorphically evaluate non-polynomial functions (i.e., `ACT`, `POOL`), we construct our CNNs differently, replacing incompatible

layers with HE-friendly alternatives. For the HE-friendly pooling layers, we use sum-pooling or scaled-mean pooling proposed by CryptoNets [28] to avoid the division in a typical average-pooling layer.

### 2.1.3 Approximating functions using polynomials

To generate HE-friendly activation functions, we use polynomial approximations of traditional activation functions. While our work primarily uses the best squares approximation method, other works in the field have also used Taylor series expansions and Best uniform approximations. All three methods are supported by the Weierstrass approximation theorem [76], Stone-Weierstrass theorem [72], and Haar theorem [35].

#### 2.1.3.1 Taylor expansion

Taylor's Theorem [43] is the most elementary approximation method in numerical analysis. It expands a $k$ times differential function into the general $k$-th order Taylor polynomial $T_n(x) = \sum_{n=0}^{k} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + R_k$ around a given point $x_0$ with $k \in \mathbb{Z}$. Then we define $\sum_{n=0}^{k} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$ as Taylor Series and $R_k = \int_{x_0}^{x} \frac{f^{(n+1)}(t)}{n!}(x - t)^n \mathrm{d}t$ as the remainder of Taylor polynomial. Taylor expansion has been widely used in computational optimization because it can convert complex mathematical expressions into Taylor polynomials using only additive and multiplicative operations [12, 33].

#### 2.1.3.2 Best uniform approximation

This approximation method is based on the $L^\infty$ norm, defined as $\|f(x)\|_\infty = \max_{x\in[a,b]} |f(x)|$, where $[a, b]$ is a region of interest as shown in Fig. 2.5(a). It aims to minimize the maximum distance between the function being approximated $f(x)$ and approximation polynomial $p(x)$. In Fig 2.5(a), we denote the error function in best uniform approximation as $\epsilon = \|f(x) - p(x)\|_\infty = \max |f(x) - g(x)|$. To generate the polynomial approximation, the following optimization problem has to solved: $\min \max |f(x)-p(x)|$. But since the problem is hard to solve, a general and practical way is to directly expand $f(x)$ into Chebyshev series [14] taken as the approximation polynomial. The Chebyshev series is a class of orthogonal polynomials defined as $T_n(x) = \cos(n \cdot \arccos(x))$ with $x \in [-1, 1]$. Given $T_0(x) = 1$ and $T_1(x) = x$, it is convenient to calculate Chebyshev polynomials: $T_{n+1}(x) = 2xT_n(x)-T_{n-1}(x)$. Therefore, we can first expand $f(x)$ into Chebyshev series in $[-1, 1]$. Then, we calculate its Chebyshev coefficients $c_k = \frac{2}{\pi} \int_{-1}^{1} f(x) \frac{T_k(x)}{\sqrt{1-x^2}}$ and adjust approximation domain from $[-1, 1]$ to $[a, b]$, for the $k$-th order Taylor polynomial. Algorithm 2 gives an elaborate description of the Chebyshev approximation.

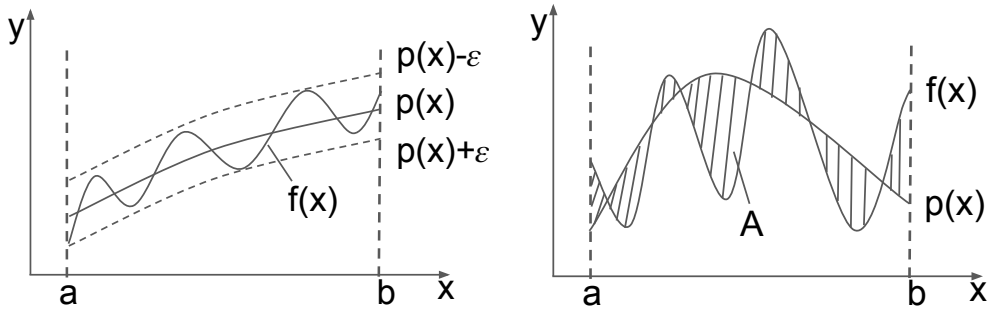
#### 2.1.3.3 Best squares approximation

Best squares approximation is based on $L^2$ norm, defined as $\|f(x)\|_2 = [\int_a^b (f(x))^2 \mathrm{d}x]^{\frac{1}{2}}$, where $[a, b]$ is the region of interest as shown in Fig. 2.5(b). Unlike best uniform approximation, this method aims to minimize the area between the function $f(x)$ being approximated and the polynomial approxi-

**Input:** $c_k$, $T_k(x)$
**Output:** $f(x)$

1: Rewrite $f(x)$ into Chebyshev Series form: $f(x) = \sum_{k=0}^{n} c_k T_k(x)$, $x \in [-1, 1]$
2: Compute Chebyshev coeff.: $c_k = \frac{2}{\pi} \int_{-1}^{1} f(x) \frac{T_k(x)}{\sqrt{1-x^2}}$
3: Convert approximation domain from $[-1, 1]$ to $[a, b]$: $x = \frac{2x'-a-b}{b-a}$

**Algorithm 2:** Best Uniform Approximation




(a) $L^\infty$ norm (b) $L^2$ norm

Figure 2.5: Error functions

mation $p(x)$. In Fig. 2.5(b), we denote A as the total area between $f(x)$ and $p(x)$ in $[a, b]$. We can also define the error function in best squares approximation as $\epsilon = \|f(x) - p(x)\|_2 = \int_a^b (f(x) - p(x))^2 \mathrm{d}x$. In our experiments, given discrete data points, we use least squares method [48] to approximate $f(x)$. In Section 4.2, the novel weighted method is an extension of least square approximation. Instead of evenly sampling $f(x)$ in a given domain, we collect points in a weighted way.

## 2.2 Related Work

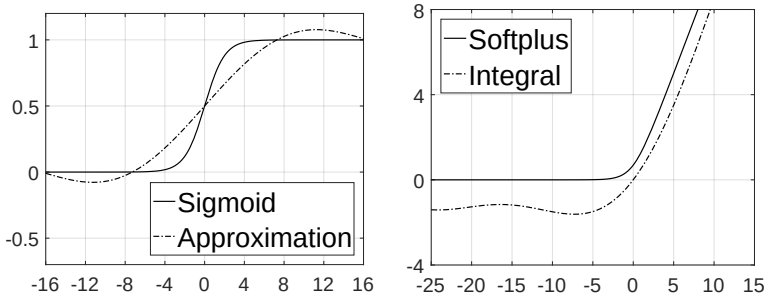
CryptoNets [28] was first to demonstrate the ability to homomorphically evaluate neural network on encrypted data for secure image classification. However, a critical flaw in the design was the properties of the power function $x^2$ used as the HE-friendly activation functions replacement (Sec. 2.1.1.3). This substitution caused instability during training, preventing its use in deeper networks. As reported by the authors, CryptoNets cannot sustain a network with more than two `CONV-ACT-POOL` layers while maintaining high efficiency and accuracy. This was attributed to the unbounded derivative of the square function $x^2$. Also, the use of sum-pooling (to avoid division operation) instead of average or max-pooling was an added impact to accuracy. As a result, CryptoNets achieved 98.95% on the MNIST handwritten digit dataset with state-of-the-art result being 99.77% on plaintext image data. Fig. 2.3(c) shows how the square function used in CryptoNets behaves in comparison to other well-performing activation functions.

After CryptoNets [28] demonstrated the ability to homomorphically evaluate neural network on encrypted data for secure image classification, several follow-up work [13, 16, 38, 77] attempted to improve classification accuracy using additional techniques and deeper networks. Chabanne *et al.* [13] attempted to approximate the ReLU activation function using Taylor series polynomials. Fig. 2.3(c) shows their best-performing approximated polynomial with degree 4; i.e., `Polyfit4`. The use of a relatively low degree polynomial allowed a low multiplicative depth, resulting in efficient evaluation

on encrypted data. However, the approximated polynomial is not accurate outside the bounds of approximation. To shrink the range of the input distribution, Chabanne *et al.* proposed to use a batch normalization (`BAT`) [40] layer before every activation layer, as illustrated in Fig. 2.1. In addition to this layer, a two-stage training process was proposed, where the network is first trained with the original activation function to achieve optimized weights. In the second stage the activation function is replaced with the approximation of the activation and the weights are fine-tuned by continuing the training at a low learning rate. Using these new methods, they improved the classification accuracy to 99.30% on the MNIST dataset.

Instead of using Taylor series approximation, Chabanne *et al.* suggested to make the coefficients of the polynomial as trainable parameters as well. Wu *et al.* [77] explored this idea and trained a model with the polynomial coefficients in each activation layer as trainable parameters. While the model achieved a classification accuracy of 99.70%, it is at the expense of optimizing a large number of parameters. More importantly, the trained model may not be widely applicable in practice because this model is too specific to the training set.

Hesamifard *et al.* proposed CryptoDL [38] to evaluate a more complex image dataset, CIFAR-10 [42], in addition to the MNIST dataset. Based on the conclusions from CryptoNets, CryptoDL focused on stabilising the training process by approximating the derivative of the original activation function. The integral of this approximation, illustrated in Fig. 2.6, was used within




(a) Approximation of the derivative (b) Integral of the approximation

Figure 2.6: Generating HE-friendly activation functions in CryptoDL

the network during training and inference. Using this approach, CryptoDL achieved a classification accuracy of 91.55% on the CIFAR-10 dataset.

Many modern deep learning problems require architectures that are significantly deeper than the networks discussed in the above approaches. Faster CryptoNets [16] presented a practical implementation of deep CNNs that can process encrypted data using transfer learning. Transfer learning is a training method that uses a network pre-trained on a related dataset and retrains the final layers on the data of the new task. Their proposed approach used a 50 layers residual network (ResNet) where only the final few layers perform inference on encrypted data; i.e., the feature maps are encrypted. The initial layers process on plaintext data. The authors tested this approach on a diabetic retinopathy dataset [34] and achieved a classification accuracy of 76.47% (Baseline ResNet scoring 80.61%).

# Chapter 3

# Analysis

## 3.1 Analysing rectified linear units as activations

To build an effective polynomial activation function, we first looked into the key factors that contribute to the performance of rectified linear units, in particular, to the ReLU activation function. Glorot *et al.* [29] suggests that using a rectified non-linearity gives rise to sparse representations within the network. This in turn has multiple benefits such as, linear separability and information disentangling of the input data. Their experiments on image data also indicate that training sessions proceed better when signals are either completely off or are linear in output.

To analyze the benefit of these properties, we construct two polynomial activations, Activation 1 and 2. The former activation closely emulates the rectified portion of ReLU for inputs $x < 0$. Activation 2 on the other hand, emulates the linear output of ReLU for inputs $x > 0$. Both activation functions replicate ReLU between [-1, 1] and pass through the origin. It must be noted that both properties cannot be incorporated simultaneously into a polynomial due to the structure of ReLU. Even with a high degree polynomial, it is not possible to efficiently approximate ReLU because the function instan-

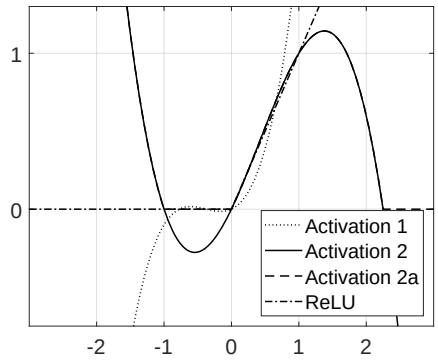



Figure 3.1: Polynomial activations approximating ReLU

taneously changes at $x = 0$. Therefore, the polynomials were approximated to the respective section of interest, *i.e.* , [-1, 0] for Activation 1 and [0, 1] for Activation 2. To understand the effect of both properties from a polynomial perspective, we also construct a piecewise Activation 2a, *i.e.* , max(0, Activation 2), which has a rectified output for $x < 0$.

The above polynomials were then tested on the MNIST dataset using the Light and Deep CNN architectures from Chabanne *et al.* [13]. Both architectures use a batch normalization layer before every activation layer to reduce the distribution of inputs. We also test the second degree polynomial from Chabanne *et al.* to compare performances.

From this experiment we make two main observations. Firstly, all polynomials approximations performed close to ReLU within the margin of error. We do not see a significant difference in performance between our polynomials with the properties of ReLU and the others.

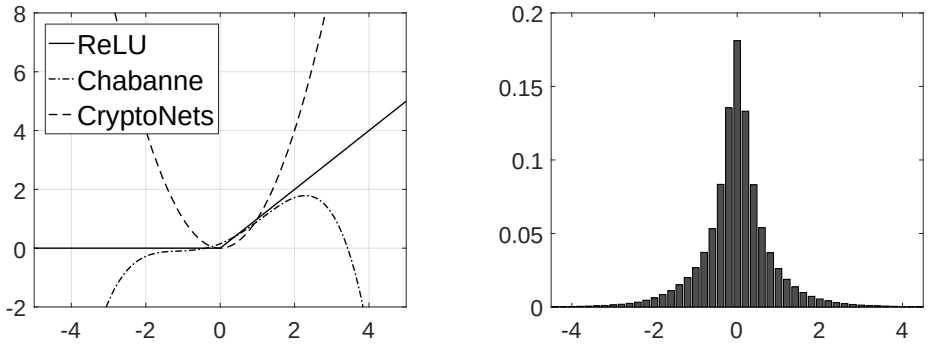
Table 3.1: Performance result of different activations on the MNIST database

| **Activation** | **MNIST Light CNN** | **MNIST Deep CNN** | **CIFAR-10** |
|---|---|---|---|
| Activation 1 | 98.25 | 99.30 | - |
| Activation 2 | 98.45 | 99.57 | - |
| Activation 2a | 97.69 | 99.14 | - |
| ch_2 | 97.93 | 99.60 | 81.00 |
| ReLU | 98.43 | 99.63 | 93.32 |

Secondly, none of our activations performed better than chance on CIFAR-10, while the second degree polynomial from Chabanne *et al.* was able to achieve 81.00%. This brought our attention to the range of approximation for each polynomial. The second degree polynomial from Chabanne *et al.* was approximated between a significantly larger range of [-3, 3] compared to our proposed polynomials. While Activation 1, 2 and 2a perform well on MNIST, they fail to learn on CIFAR-10. Therefore, our next step was to analyse the effect of the dataset on the distribution of inputs to the activation layer.

## 3.2 Analysing inputs to the activation function

In the previous section, we show how our proposed activations perform well enough on the MNIST dataset, but not for CIFAR-10, a significantly more complex dataset. However, we also saw another polynomial in the test successfully learn in the training process, albeit with a lower score. All polynomials were approximating the same activation function, ReLU, but within different ranges. From Table 3.1 our activations that did not learn mirrored ReLU between the range [-1, 1] but the other polynomial mirrored ReLU be-




(a) Growth of polynomial approximations and power functions

(b) A typical output distribution of a batch normalization layer

Figure 3.2: Polynomial approximation of ReLU

tween [-3, 3]. Therefore, we analysed the the inputs to the activation function and report our findings in this section.

Initially proposed by Chabanne *et al.* , networks with HE-friendly polynomial activations have been trained with a batch normalization layer before every activation layer. The BN layer transforms the data distribution to have zero mean and unit variance. Essentially, this results in a distribution with close to 99% of the data between the range [-3, 3] as shown in Fig. 3.2(b).

Table 3.2: BN Output distribution characteristics for different datasets

| **Dataset** | **Max. Std. Deviation** | **Max. Distribution Range** |
|---|---|---|
| MNIST | 1.20 | [-8.44, 8.51] |
| Fashion MNIST | 1.12 | [-15.35, 20.10] |
| CIFAR-10 | 1.09 | [-30.17, 21.31] |

However, upon analyzing the outputs from the Batch normalization layer, we observe that the properties of the output distributions are dataset

dependent. While the structures of these distributions are similar, the range of distributions widely vary. Depending on the complexity of the dataset, we observe that at least 98% of the data lies between the range [-3, 3], but the ranges can be as large as [-30, 21]. Table 3.2 lists the maximum observed range and standard deviations of the output distributions across different datasets.

Based on the results seen in Table 3.2, we can understand why our proposed activations in Section 3.1, Activation 1, 2 and 2a, perform poorly on the CIFAR-10 dataset. The maximum observed range for the dataset is significantly wider than the range between which the polynomial replicated the ReLU activation function. When the polynomial activation receives an input well outside the range of approximation, the outputs are very large resulting in errors propagating across layers. Moreover, during the backpropagation process, the gradients explode and hence no learning takes place.

We now pose the question – Will a polynomial with an approximation range larger than [-3, 3] be more suited for a dataset like CIFAR-10? To answer this question we construct two polynomials of degree 4 between the bounds [-7, 7] and [-25, 25]. In this experiment we also compare the performance with a degree 4 polynomial approximated between [-3, 3].

Upon testing the three polynomials on the CIFAR-10 dataset, our first observation was the significant increase in accuracy for the polynomial approximated between [-7, 7] compared to the range [-3, 3]. By approximating between a larger range, the activation layer is able to process more inputs accurately and hence have a more stable training process.

Table 3.3: Performance of CNNs with polynomial activations between different ranges

| **Range** | **Accuracy** | **Approximation error in [-3, 3]** |
|---|---|---|
| [-3, 3] | 81.32 | 4.187 |
| **[-7, 7]** | **88.25** | **4.508** |
| [-25, 25] | 83.45 | 5.392 |

On the other hand, approximating between a too large a range seems to negatively affect the training process. This is because a large range of approximation sacrifices the quality of the approximation. Table 3.3 lists the results of the test along with the error of approximation between the range [-3, 3]. We chose to report within this range because most of the data lies within this range due to the BN layer and hence it is vital that the polynomial closely approximates this region. We can see that the error is highest for [-25, 25] and lowest for [-3, 3]. But at the same time, the [-7, 7] range yields the best result even with a relatively higher error of approximation than [-3, 3].

From our experiments in this section, we can conclude that it is necessary to strike a balance between maintaining an acceptable error of approximation between [-3, 3], while covering a range larger than [-3, 3]. This is true, especially for complex datasets that have large input distributions to the activation layer.

## 3.3 Analysing effective activation functions

Related work in the field have achieved good performance on datasets like MNIST by only approximating the structure of the polynomial. However, from our experiments in Section 3.1 we observe that merely mimicking the structure of an activation function does not yield effective performance. Instead, we choose to list and analyse the properties generally observed in effective classical activation functions like ReLU, Tanh, and Sigmoid. Incorporating these properties while constructing HE-friendly polynomials can form the basis of a systematic approach.

### 3.3.1 Non-linear

As discussed in Section 2.1.1, the primarily role of the activation layer is to capture the non-linearity of complex features in CNNs. Naturally, many effective activation functions are non-linear. A linear function $f(x)$ is one which satisfy both of the following properties: $f(x + y) = f(x) + f(y)$ and $f(\alpha x) = \alpha f(x)$. Nonlinear functions are those who do not follow the above definition. Complex tasks such as classification of images or speech involves the separation of non-linear data and can be performed well only using non-linear models. Non-linear activation functions such as ReLU and Sigmoids enable neural networks with the capability to perform this task. This is because neural networks with effective non-linear activation functions are universal function approximators [19]. Every complex task can be abstracted as a function that maps an input to an output. Without the use of such activation functions, the

networks would essentially be a linear model.

### 3.3.2 Differentiable

Backpropagation is the most popular and effective training method that has been used for neural networks. This training method makes use of the derivatives of the functions used in the network to adjust the weights while optimizing errors. Due to this reason, it is necessary for all components in the network, including the activation function, to be differentiable.

A function $f$ is said to be differentiable if the derivative $f'(x)$ exists. This property ensures that the derivative is defined and exists at every value. The functions Softplus, Sigmoid and Tanh and their derivatives are differentiable over the entire domain. However, effective activation functions including ReLU and Leaky ReLU are only piecewise differentiable because they are piecewise functions and do not have derivatives at the origin point. In practical applications of backpropagation, we just set the value of the derivative of ReLU at the origin point as zero.

### 3.3.3 Continuous

A function $f(x)$ is said to be continuous at a given point $x_0$ if,

$$\lim_{x \to x_0} f(x) = f(x_0)$$

*i.e.* sufficiently small changes in the inputs to the function result in arbitrarily small changes in the outputs. Effective activation such as ReLU, Softplus, and Tanh are continuous. Polynomials are also a classic class of continuous

functions over the domain $\mathcal{R}$ and linear combination of polynomials are also continuous.

### 3.3.4 Zero-centered

It has been long known that neural networks can learn faster if activation functions in hidden layers are centered around zero [45, 68]. LeCun *et al.* [45] gave a strict proof to the zero-centered property of effective activation functions. In this section, we provided a more intuitive interpretation to explain this observation.

We first denote $\vec{x} = (x_1, x_2, ..., x_n)$ as a n-dimensional input vector and $\vec{w}_1 = (w_1, w_2, ..., w_n)$ as an initial weight vector. Then with given threshold $b$, we can define a neuron model as an input-output map $f(\vec{x}; \vec{w}, b)$, satisfying $f(z) = f(\sum_{i=1}^{n} w_i x_i + b)$. The gradient descent process of each parameter $w_i \in \vec{w}$ based on the given cost function $L(\vec{x})$ and learning rate $\eta$, can be shown as $w_{i+1} = w_i - \eta \frac{\partial L}{\partial w_i} = w_i - \eta \frac{\partial L}{\partial f} \frac{\partial f}{\partial z} x_i$. Hence, the renewal equation above shows that the updated direction of parameter $w_i$ is entirely based on the value of $x_i$ because parameters including $\eta$, $\frac{\partial L}{\partial w_i}$ and $\frac{\partial f}{\partial z}$ can all be seen as constant term.

To explain the zero-centered property, we further assumed that the optimal weight vector $\vec{w}^\star = (w_1^\star, w_2^\star, ..., w_n^\star) \neq \vec{w}_1$ and the previous neuron unit takes Sigmoid $g(x) = \frac{1}{1+e^{-x}}$, a typical nonzero-centered function, as activation functions. Then, due to $g(x) \in (0, 1)$, we can know that $x_i > 0$ for each input value $x_i \in \vec{x}, i = 1, ..., n$, which will cause the renewal process of each $w_i \in \vec{w}_1$

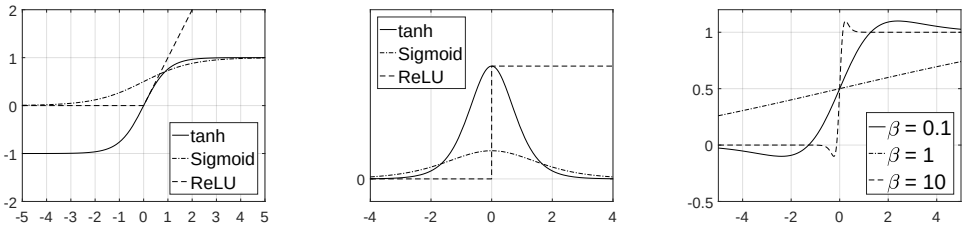
to take a z-path from the initial weight vector $w_1$ to the optimal weight vector $w^\star$ and the learning speed of neural networks will be much slower.

### 3.3.5 Monotonic

A function $f$ is monotonic when it is either increasing or decreasing throughout the entire domain of inputs, *i.e.* the function always grows in a single direction. During training, a neurons weight might be changed to increase or decrease its influence on neurons in the next layer. A monotonic activation function has shown to make this behaviour predictable and thus enabling faster optimization. Using a non-monotonic activation function might have the opposite effect because of how non-monotonic functions change in direction of growth. Most of the effective activation functions such as ReLU, Sigmoid, Tanh and Softplus are monotonic in nature. However, it is important to note that neural networks with non-monotonic activation functions can be optimized as well – the network might require a longer training time. For example, non-monotonic trigonometric functions such as the periodic sine wave have been used as activation functions to train neural networks successfully [61].

### 3.3.6 Bounded derivative

It has been observed that activation functions with bounded derivatives contribute to effective performance in neural networks. This is because a bounded derivative restricts the training algorithm from making large updates




(a) Effective activations used in Deep Learning (b) Bounded derivatives of effective activations (c) Bounded derivatives of Swish for different $\beta$

Figure 3.3: Effective activation functions and their (bounded) derivatives

to the weights of the network [54]. Not preventing this phenomenon could lead to large fluctuations in the network weights and create instability during the training process. We clearly see this issue in our experiments in Section 3.1 for Activations 1, 2, and 2a.

The same property can be seen in well-performing activation functions discovered by Ramachandran *et al.* [64]. The research used a reinforcement learning technique to find effective activation functions from a search space of functions. The search space was composed of a combination of linear and non-linear functions such as $x^2$, $sin(x)$, $x_1 + x_2$. Every well-performing activation discovered using this process has a bounded derivative. The most effective activation discovered was named Swish. Defined as $x \cdot sigmoid(\beta x)$, Fig. 3.3(c) shows how the derivative of Swish stays bounded with different values of $\beta$, which the network can adjust as a trainable parameter. Figure 3.3(b) shows the bounded derivatives of the effective activation functions ReLU, Sigmoid and Tanh.

Multiple similarities can be observed between classical activation functions and polynomials based on the above properties. Like other well-known activations, polynomials are differentiable, non-linear and continuous. However, unlike most traditional activations, polynomials are not monotonic and do not have a bounded derivative. Out of these two dissimilarities, we suspect the lack of a bounded derivative to affect the performance of the polynomial adversely more than the lack of monotonicity. Recall that polynomial approximations are accurate only within their range of approximation. Due to this reason, any input outside this range causes error to propagate forward and significantly hamper learning during the backpropagation of the gradients.

However, we can also consider a polynomial to be locally bounded by approximating it between the range of expected inputs. The batch normalization layer before every activation helps us in this regard by reducing the range of the input distribution.

# Chapter 4

# Proposed Solution

From our analyses on how the batch normalization layer transforms data and what factors are key contributors to a good activation function, we propose the following solutions to generate efficient HE-friendly polynomial approximations. A key contributor to the performance of the activation function is having a bounded derivative. However, since the polynomial approximation will be accurate only within their approximation range, it is necessary to restrict the range of inputs using batch normalization layer. While proposing these methods, we keep in mind the effect of complex datasets on the input distribution to the activation layer and also take advantage of the nature of the distribution.

## 4.1 Training with multiple polynomial activations

The experiments in Section 3.2 show us that it is important to approximate a polynomial between a range larger than [-3, 3]. Approximating between a larger range will result in a higher error of approximation in the critical region of [-3, 3] and can negatively affect training as seen in Table 3.3. However, based on our analysis of the BN outputs on the network, we see that

the range of inputs differs from layer to layer. For example, Table 4.1 lists the maximum observed range per layer for a network trained on CIFAR-10 using Softplus.

Table 4.1: Input ranges recorded for each activation layer after training on CIFAR-10 using softplus

| **Activation layer** | **Max. range per layer** |
|---|---|
| 1 | [-22.04, 21.31] |
| 2 | [-30.17, 18.85] |
| 3 | [-25.73, 10.62] |
| 4 | [-12.81, 9.34] |
| 5 | [-11.76, 8.75] |
| 6 | [-12.98, 10.54] |
| 7 | [-14.26, 10.70] |
| 8 | [-15.72, 18.08] |
| 9 | [-2.83, 7.36] |

We can take advantage of this phenomenon by approximating a polynomial for every layer. The benefit of this approach is two-fold. Firstly, the approximation for every layer would be able to accept most of the inputs while having the lowest error of approximation. This way, layers with a smaller input range will not be constrained by a single polynomial catering to the largest range observed.

The second benefit to this approach is that each layer can use a polynomial with a suitable degree to balance between the error of approximation and the number of multiplications. Recall that due to the properties of Homomorphic Encryption, we are constrained by the number of multiplications that can be performed. In order to keep the network HE-friendly, we must use

low-degree polynomials to maintain an acceptable multiplicative depth. Using a multi-polynomial approach can help us in this regard by permitting us to use lower degree polynomials for smaller ranges and higher degree polynomials for larger ranges. This combination will result in a lower multiplicative depth than using a single high degree polynomial for all layers.

## 4.2 Weighted Polynomial approximations

From the experiments in Section 3.2 we observe that more than 98% of the inputs to the activation layer are between [-3, 3]. At the same time, we also show that it is necessary to approximate polynomial replacements between a range larger than [-3, 3], particular in the case of complex datasets. Approximating between a larger range however, can increase the error of approximation between [-3, 3] and can have a negative impact on performance as shown by our experiments before. To approximate a polynomial within a larger range without compromising the error of approximation, we propose a weighted approximation technique.

Formally, to approximate a function $f$ between a range $[L_{low}, L_{high}]$, we first sample $f$ as follows:

$$Y = \{f(x) : x \in X\} \tag{4.1}$$

where $X = \{L_{low}, \ldots, L_{high}\}$ is the set of linearly spaced points. Figure 4.1(a) visually depicts this process with 100 sample points. Then, using a polynomial regression function like *polyfit* from *MATLAB*, a polynomial of degree $n$ is fit

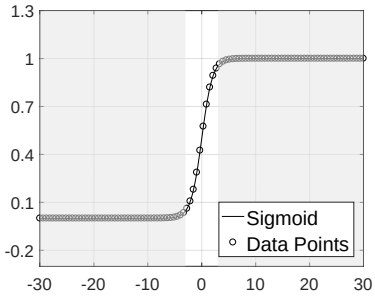
to $X$ and $Y$ using an error measure such as least-squares.

We propose a weighted approximation technique to maintain an acceptable error of approximation between [-3, 3] in large range approximations. Our method takes advantage of the input data structure where at least 98% of data lies between the range [-3, 3]. Instead of sampling the function linearly, we sample the range [-3, 3] at a higher rate than the remaining sections using a weight. More formally, the activation function $f$ is sampled between the range $[L_{low}, L_{high}]$ where

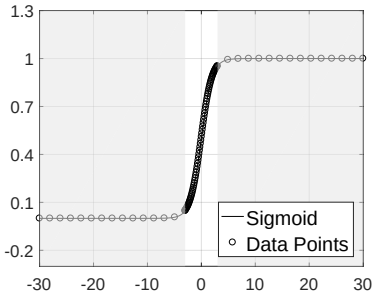
- $X = [X_1\ X_2\ X_3]$
- $X_1 = \{L_{low}, \ldots, -3\}$ containing $\frac{(1-R)n}{2}$ linearly spaced points,
- $X_2 = \{-3, \ldots, 3\}$ containing $Rn$ linearly spaced points
- $X_3 = \{3, \ldots, L_{high}\}$ containing $\frac{(1-R)n}{2}$ linearly spaced points,
- $R \in [0, 1]$ is the weight determining the rate of sampling between [-3, 3].

The sampling rates for each region in the method proposed above can be calculated as follows:

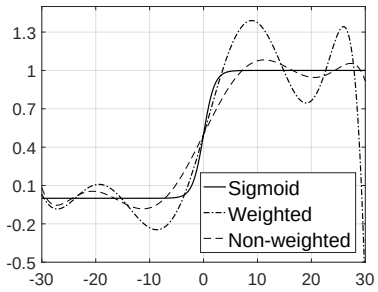
- $\frac{6}{Rn}$ for the region $X_2$
- $\frac{2}{n}\frac{L_{high}-3}{1-R}$ for regions $X_1$ and $X_3$, given $|L_{high}| = |L_{low}|$




(a) Conventional method of linearly sampling a function (non-weighted)




(b) Proposed weighted sampling of activation function




(c) Degree 8 polynomial approximations of Sigmoid using non-weighted and weighted sampling

Figure 4.1: Improving quality of approximations using our proposed method

Figure 4.1(b) visualizes this sampling approach with $n = 100$, $R = 0.7$, $L_1 =$ -30 and $L_2 = 30$. We have chosen to visualize the sampling on the Sigmoid activation function because the difference is visually more apparent than ReLU or Softplus. However, the effect of the approach will be the same for any other activation function. Using *polyfit* with the above sampling rates for each region specifically minimizes the error between [-3, 3] while approximating through a large range. The benefit of this approach over using a linear sample of points

is the larger range covered for the same error of approximation between [-3, 3].

One thing to note in our proposed approach is the expense of weighting the approximation towards [-3, 3]. Due to a higher weight in this section, the region of approximation outside [-3, 3] would have a high error of approximation. However, due to the sparsity of inputs in this region, it should not adversely affect the performance of the model.

## 4.3 Discovering the optimal range for a dataset

In Section 3.2 we realize that it is beneficial to approximate polynomials beyond [-3, 3]. But there is clearly a range beyond which the performance of the model starts dropping. There exists some range beyond [-3, 3] where the network would perform most optimally. It is also necessary to observe the consistency of this behaviour across different settings. To understand where the optimal point lies, we perform a grid search across multiple factors listed below:

- Method of polynomial approximations
- High and low degree of approximations
- Type of activation function approximated
- Dataset for the CNN
- Structure of the CNN

For this grid search, we approximate polynomials of degree 4 and 7 using three approximation methods:

- Fitting a polynomial to a linear set of points using the least-squares approach used by Chabanne *et al.*
- Chebyshev polynomial approximations proposed by Hesamifard *et al.* We use the measure $d\mu = e^{(\frac{-1}{1e-5+x^2})}$ specified in CryptoDL to approximate the derivative of the activation. The integral of the generated approximation is then used as the polynomial activation in the network.
- Our proposed weighted polynomial approximation with $R = 0.7$

Multiple polynomial candidates are approximated between the ranges of $[-L, L]$ where $L \in \{3, 5, 7, \ldots N\}$ and $N$ is the observed absolute maximum in the BN output distribution for that dataset 3.2. We also compare the performance difference between approximations on ReLU and Softplus. The tests will be conducted on the MNIST, FMNIST and CIFAR-10 dataset. Since the CIFAR-10 uses a different architecture than the other datasets, the effect of a different architecture will also be noted in our experiments.

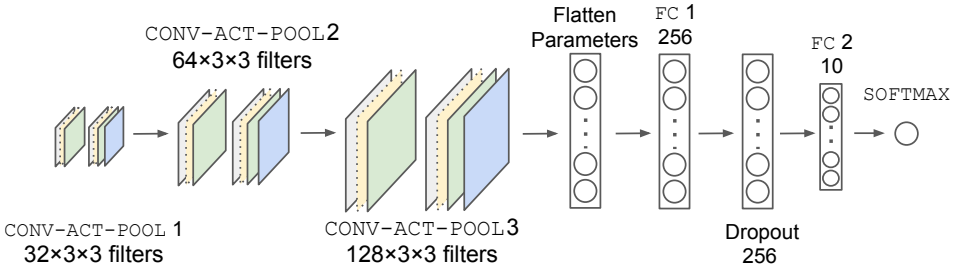
# Chapter 5

# Evaluation and Results

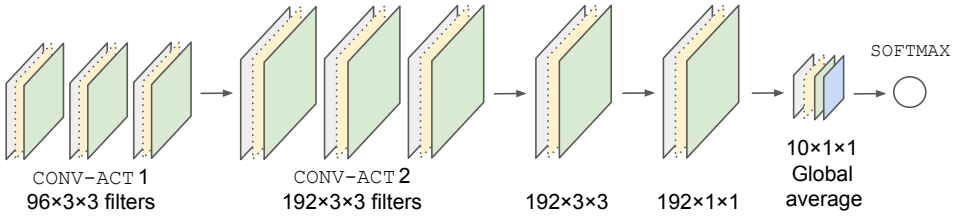
## 5.1 Network architecture and training procedure

We adapt the Deep CNN model proposed by Chabanne *et al.* to test our proposed solutions on the MNIST and FMNIST datasets. Figure 5.1(a) shows an overview of the model with 14 layers and 6 activation layers. Following is the network configuration:

- Block 1 contains two convolutional layers with with 32 filters of size 3 x 3 followed by an average pool.
- Block 2 and block 3 follow are similar to block 1 but have 64 and 128 filters of size 3 x 3 for the convolutional layers respectively.
- Following these convolutional blocks is a fully connected layer with 256 neurons.
- After a dropout layer with $p$=0.5, the network results are parsed from a final fully connected layer of 10 neurons.
- All convolutional layers are followed by a batch normalization layer which is followed by an activation layer.




(a) Architecture for MNIST and FMNIST




(b) Architecture for CIFAR-10 (uses stride=2 in place of pooling layer)

Figure 5.1: CNN architectures with HE-friendly pooling and activation layers

We use the architecture proposed by Springenberg *et al.* [71] for CIFAR-10, shown in Fig. 5.1(b). Unlike the conventional CNNs, this architecture does not use pooling layers. Instead it uses convolutional layers with stride = 2 to reduce the output size of the feature maps, just as how a 2x2 pooling layer would. This network contains 18 layers with 9 activation layers:

- Block 1 contains three convolutional layers with 96 filters of size 3 x 3. To emulate a pooling layer, the final convolutional layer has a stride = 2 which reduces the resolution of the outputs by half.
- Similar to block 1, block 2 also contains 3 convolutional layers but with 192 filters of size 3 x 3. The final convolutional layer has stride = 2.
- The next block contains a single convolutional layer with 192 filters of

size 3 x 3. This is followed by another convolutional layer with 1 x 1 convolutions.

- The final convolutional layer reduces the number of outputs by only containing 10 filters. A final global average pooling function gathers the results of the network.

For both models, we use the training procedure proposed by Chabanne *et al.* The model is initially trained using a traditional activation function like Softplus. The activation function is then replaced with its polynomial approximation, and the model is further trained with the pre-trained weights as a starting point. The training on MNIST using the original activation is conducted for 350 epochs with an initial learning rate of 0.01. Once the activations have been replaced, the training continues at $lr = 10^{-6}$ for 200 epochs.

For CIFAR-10, the first phase of training is performed for 450 epochs with a learning rate scheduler scaling the learning rate by 0.1 at epochs 300, 350 and 400. After replacing the activation function with an approximation, the training continues at $lr = 10^{-6}$ for 300 epochs with the scheduler scaling the learning rate by 0.1 at epochs 150, 200 and 250.

The time taken to train a model using our method is significantly higher (around 2.5 times) because of the two stage training process. However, this is a small price to pay for an effective network with a polynomial activation.

Because our research focuses on devising a method to generate effective polynomial activations, all our experiments and results are only on plaintext

data.

## 5.2 Experiment Setup

For our experiments we used a system with a Xeon Silver 4114 CPU at 2.20 GHz with 192 GB of RAM and an NVIDIA Tesla P100 GPU running Ubuntu 16.04. All CNN models were constructed and run using the PyTorch 1.02 library. We used MATLAB's *polyfit* to generate polynomial approximations for both the weighted and non-weighted approaches. For Chebyshev approximations, we used the PyProximation library used by Hesamifard *et al.*

## 5.3 Results

In this section, we will describe the results of our tests using the proposed approximation methods in Chapter 4. To understand the performance characteristics in detail, we would be comparing against existing work and will be testing on multiple polynomial approximation and datasets. Based on the results obtained, we list our observations and conduct further analysis.

### 5.3.1 Multi-polynomial setup

We first tested our multi-polynomial approach separately by analysing the inputs to the activation functions and noting the range for each activation layer – Table 4.1 shows the ranges obtained for a network trained on CIFAR-10 using the Softplus activation function. For every layer, a polynomial approximation of Softplus is constructed between the corresponding

range and a family of polynomials is built. As explained in Section 5.1, we then replaced the activation layers with the respective polynomial continued the training process. Using this training approach, we were able to achieve a marginally higher performance than using a single polynomial, with the model correctly classifying 90.38% of the data correctly on the CIFAR-10 test set. While the improvement is only marginal, the result empirically shows that our hypothesis holds and that the network can perform better with a family of polynomials.

Table 5.1: Performance of models using the approximation method from [38] and training method from [13]

| **Activation** | **Accuracy** |
|---|---|
| Single polynomial | 89.34% |
| Multi polynomial | 90.38% |
| Softplus | 92.95% |

However, it must be noted that the model is extremely unstable to train using this approach. During the training process, the model would lose training stability and the performance fall to chance. To analyse the cause of this behaviour, we trained another network instance layer by layer. Once the weights of a layer adjusted to the polynomial activation, we would freeze all the weights before it, replace the next activation layer and continue training. During this process, we also analysed the inputs to every activation function and noticed a change in the distribution. After replacing an activation layer with a polynomial, the distribution of subsequent layers after training have an

increased range and standard deviation. Due to this change, the inputs in the later layers of the network fall out of the range of approximation and lead to the same gradient explosion problem.

This effect is particularly noticeable in the last few activation layers and is consistent with the observations made by Chabanne *et al.* To counter this effect, Chabanne *et al.* approximated a polynomial to with respect to the corresponding layer distribution. Unlike the MNIST dataset however, we observe significantly larger input distribution ranges for CIFAR-10. Therefore this prevents us from using the solution provided by Chabanne *et al.* Moreover, implementing this solution in our case would result in the use of relatively higher degree polynomials which counter the HE constraints that we have. Due to the difficulty in training, we decided not to include this approach while searching for an optimal range in Section 5.3.3.

### 5.3.2 Weighted polynomial approximations

To test the effectiveness of our proposed weighted polynomial approximations, we approximate three polynomials with different properties. One polynomial is approximated between [-5, 5] using the traditional approach. The remaining are approximated using our proposed approach but with different ranges, [-5, 5] and [-11, 11]. All polynomials are approximations of Softplus and were tested on the CIFAR-10 dataset.

From Table 5.2, we can see the benefits of using weighted approximations on the CIFAR-10 dataset. Compared to the conventional approach, the

Table 5.2: Comparison of approximation error and ranges according to type of approximation

| Range | Approximation method | Error in [-3, 3] | Accuracy |
|---|---|---|---|
| [-5, 5] | Non-weighted | 4.376 | 82.17 |
| [-5, 5] | Weighted | **4.354** | 85.81 |
| [-11, 11] | Weighted | 4.785 | **89.91** |

reduced error between [-5, 5] of the weighted approximation helps improve classification accuracy. As an added benefit to our approach, larger approximation ranges can be accommodated which improves training optimization, consequently improving performance. In the following experiments, we will compare weighted approximations along with other methods used in the related work.

### 5.3.3 Searching for the optimal point

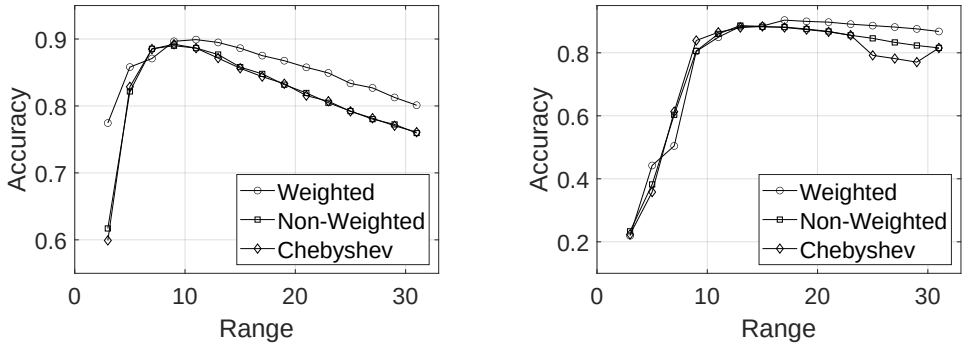
To understand where the balance between the error of approximation in of the polynomial and range lies, we perform a grid search across multiple variables that are common in CNNs. As explained in 5.1, every network instance uses the pre-trained weights of the corresponding activation as a starting point. However, due to the large number of tests in this grid search, we restrict the number of epochs for the second stage of training to 50.

To recap from section 4.3, we search across the following parameters to find where the optimal point lies:

- Method of polynomial approximations

- High and low degree of approximations
- Type of activation function approximated
- Dataset for the CNN
- Structure of the CNN

Due to the training time and difficulty, we choose not to include our Multi-polynomial approach in this analysis.




(a) Degree 4 approximation results

(b) Degree 7 approximation results

Figure 5.2: Performance of polynomial approximations using different approximation methods and at different ranges

In our first set of experiments, we test across multiple candidates to observe the difference between ReLU and Softplus polynomial approximations. As mentioned in section 4.3, we test polynomial approximations of degree 4 and 7 using three approximation methods:

- Fitting a polynomial to a linear set of points using the least-squares approach used by Chabanne *et al.*

- Chebyshev polynomial approximations proposed by Hesamifard *et al.* We use the measure $d\mu = e^{(\frac{-1}{1e-5+x^2})}$ specified in CryptoDL to approximate the derivative of the activation. The integral of the generated approximation is then used as the polynomial activation in the network.

- Our proposed weighted polynomial approximation with $R = 0.7$

We approximated 15 candidates of both Softplus and ReLU between the ranges [-3, 3] and [-31, 31]. In total, we train 180 network instances on CIFAR-10 with the epochs in phase 2 restricted to 50. Figure 5.2 shows the performance achieved using each candidate against every corresponding range and approximation method. We considered testing approximations of Sigmoid and Tanh along with ReLU and Softplus. However, as mentioned in Section 3.3, these activation functions suffer from the vanishing gradient problem. For this reason, they perform significantly worse than ReLU and Softplus and hence, we choose not to explore their approximations.

We conduct another set of experiments to examine if our observations are consistent with different datasets and network structures. To reduce the number of training instances, we test only using approximations the Softplus activation function using our proposed weighted approximation technique. These candidates are tested on three datasets, MNIST, FMNIST and CIFAR-10 using different network configurations described in Section 5.1. We summarize the results performed on CIFAR-10 in Table 5.3.

From we can clearly see that the optimal range for all approximation

Table 5.3: Optimal approximation range for different degrees and datasets

| **Dataset** | **Degree** | **Optimal range** | **Accuracy** | **Error in [-3, 3]** |
|---|---|---|---|---|
| CIFAR | 4 | [-11, 11] | 89.91 | 4.785 |
| | 6 | [-17, 17] | 90.02 | 4.740 |
| | 8 | [-21, 21] | 90.82 | 4.612 |
| | 10 | [-23, 23] | 91.55 | 4.535 |
| FMNIST | 4 | [-4, 4] | 97.84 | 4.267 |
| | 6 | [-7, 7] | 97.80 | 4.303 |
| | 8 | [-11, 11] | 97.84 | 4.345 |
| | 10 | [-14, 14] | 97.81 | 4.339 |
| MNIST | 4 | [-4, 4] | 99.59 | 4.267 |
| | 6 | [-6, 6] | 99.57 | 4.254 |
| | 8 | [-7, 7] | 99.59 | 4.213 |
| | 10 | [-9, 9] | 99.57 | 4.215 |

methods lie beyond [-3, 3]. Even with a higher error of approximation, candidates that cover a larger range of inputs perform better on CIFAR-10 regardless of the approximation method, network structure, dataset or base activation function. From these experiments, we understand that solely minimizing the error of approximation between [-3, 3] does not yield the best performing candidates. Even though more than 98% of data lies between the range of [-3, 3], it is necessary to take into account the range of the input distribution.

On the other hand, the performance starts to change after the optimal range of approximation, especially for lower degree candidates. Beyond the optimal range, the incentive to cover larger ranges is lost because the error of approximation between [-3, 3] increases beyond an acceptable level. Table 5.2 lists error of approximations between [-3, 3] for the candidates at each optimal

range.

We also observe that our proposed weighted approximation technique performed better than the other compared methods. By forcing the approximation to weight the region between [-3, 3] over the remaining region, this approximation method achieves lower error of approximations while covering larger ranges. For example, Chabanne *et al.* achieved an accuracy of 97.91% using a degree 4 polynomial activation approximated between [-3, 3]. At the optimal point, degree 4 weighted approximations were able to achieve a 99.59% classification accuracy on the MNIST dataset.

While the Softplus activation performs slightly worse than ReLU, we observe that approximations of Softplus perform significantly better than the approximations of ReLU, regardless of the approximation method. This is because the approximations are able to fit the smooth curve of Softplus better than the sudden change in the slope of ReLU at $x = 0$. Candidates generated by our technique yield better performance because they cover a larger range of approximation while maintaining an acceptable level of error as seen in Fig. 5.2. This is especially observed in the more complex datasets, FMNIST and CIFAR-10, because they have a larger input distribution to the activation function.

Based on the results obtained above, we turn our attention towards the error of approximation between [-3, 3] to understand if it has any correlation with the optimal range observed. But the correlation does not seem to be consistent enough across datasets to make a conclusion. For example, we see

a negative correlation between the optimal range and the corresponding error between [-3, 3] for CIFAR-10 and MNIST. However, for the FMNIST dataset, the effect seems to be opposite.

Moreover, these values are specific to approximations of the Softplus activation function. The relationship between the two variables for a different base activation would be dependent on its structure and might grow in a similar fashion. Instead, these ranges could be used as a guideline to reduce the size of the grid search for different setup.

# Chapter 6

# Conclusion and Future Work

In this thesis, we introduced two new improved approximation approaches to generate HE-friendly activation functions using polynomial approximations of Softplus – training and evaluating using multiple polynomials and a new weighted approximation technique. While the former yields a slight improvement and is hard to achieve, we show the latter proposed method outperforms other methods and is robust regardless of the approximation method, degree, dataset, or activation function approximated.

To improve the effectiveness of our polynomial approximations, we analyze and list multiple properties that are key contributors to performance in classical activation functions. From these properties, we show the important of a bounded derivative and how using a batch normalization layer helps us emulate this behaviour in polynomials. We also analyzed the batch normalization layer and observed the change in behaviour based on the size and complexity of the dataset. Finally, we use the structural cue of at least 98% of data residing between [-3, 3] of the batch normalization outputs to develop our weighted polynomial approximation technique. Contrary to the conventional belief, our experiment results empirically show that merely mimicking the structure of

the activation functions is not enough and that it is necessary to approximate between bounds larger than [-3, 3]. Our method performs better because it forces the approximation to weight the region between [-3, 3] over the remaining region, this approximation method achieves lower error of approximations while covering larger ranges.

There are multiple ways that our work can be augmented. The performance of our polynomial approximations are limited by performance achieved by the activation function being approximated, *e.g.* Softplus. Therefore, an improvement in classification accuracy can be achieved by first improving the performance of the network using Softplus by either changing the network structure or tuning the hyperparameters. Additionally, to reduce the instability while training using multiple polynomials, the weights of the network could be fine-tuned by training each layer sequentially. In this process, we would start with replacing only the first activation layer and fine-tune the weights. After convergence, the weights of the first layer and earlier could be frozen and the next layers can be trained in the similar way, one by one. To improve both our proposed methods, it may also be desirable to convert the coefficient of the polynomial activations to trainable parameters in the last few epochs of training.

# Bibliography


[1] Summary of the HIPAA Security Rule. `https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html`. [Online; accessed 21-Sept-2019].

[2] HElib: An implementation of homomorphic encryption. `https://github.com/homenc/HElib`, June 2020.

[3] PALISADE Lattice Cryptography Library (release 1.7.4). `https://palisade-crypto.org/`, January 2020.

[4] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)*, 51(4):79, 2018.

[5] Noreen Sher Akbar, Abdelhalim Ebaid, and ZH Khan. Numerical analysis of magnetic field effects on eyring-powell fluid flow towards a stretching sheet. *Journal of Magnetism and Magnetic Materials*, 382:355–358, 2015.

[6] Asma Aloufi, Peizhao Hu, Yongsoo Song, and Kristen Lauter. Computing blindfolded on data homomorphically encrypted under multiple keys: An extended survey. *arXiv preprint arXiv:2007.09270*, 2020.

[7] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard

learning problems. In *Advances in Cryptology – CRYPTO*, pages 595–618. Springer, 2009.

[8] Émile Borel, Paul Painlevé, and Henri Léon Lebesgue. *Leçons sur les fonctions de variables réelles et les développements en séries de polynomes: professées à l'École normale supérieure*. Gauthier-Villars, 1905.

[9] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science Conference (ITCS)*, pages 309–325. ACM, 2012.

[11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[12] COE Burg and JC Newman Iii. Computationally efficient, numerically exact design space derivatives via the complex taylor's series expansion method. *Computers & Fluids*, 32(3):373–383, 2003.

[13] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive*, 2017:35, 2017.

[14] Pafnutiĭ Lvovich Chebyshev. *Théorie des mécanismes connus sous le nom de parallélogrammes*. Imprimerie de l'Académie impériale des sciences, 1853.

[15] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[16] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018.

[17] Jack LH Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. Doing Real Work with FHE: The Case of Logistic Regression. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 1–12, New York, NY, USA, 2018. ACM.

[18] W Cui, KA Gawecka, DM Potts, DMG Taborda, and L Zdravković. Numerical analysis of coupled thermo-hydraulic problems in geotechnical engineering. *Geomechanics for Energy and the Environment*, 6:22–34, 2016.

[19] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[20] Bhaskar DasGupta and Georg Schnitger. The power of approximating: a comparison of activation functions. In *Advances in neural information processing systems*, pages 615–622, 1993.

[21] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[22] R Ellahi and M Hameed. Numerical analysis of steady non-newtonian flows with heat transfer analysis, mhd and nonlinear slip effects. *International Journal of Numerical Methods for Heat & Fluid Flow*, 22(1):24–38, 2012.

[23] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. https://eprint.iacr.org/2012/144/20120322:031216, March 2012.

[24] Emily Flitter and Karen Weise. Capital One Data Breach Compromises Data of Over 100 Million. `https://www.nytimes.com/2019/07/29/business/capital-one-data-breach-hacked.html`. [Online; accessed 21-Sept-2019].

[25] Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, 2007:15, 2007.

[26] J Garcia-Margallo, J Diaz Bejarano, and S Bravo Yuste. Generalized fourier series for the study of limit cycles. *Journal of Sound and Vibration*, 125(1):13–21, 1988.

[27] Craig Gentry and Dan Boneh. *A fully homomorphic encryption scheme*, volume 20. Stanford University Stanford, 2009.

[28] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

[29] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[30] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[31] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. of Computer and System Sciences*, 28(2):270–299, 1984.

[32] Andy Greenberg. Complete guide to gdpr compliance. `https://gdpr.eu/`. [Online; accessed 21-Sept-2019].

[33] Andreas Griewank, Jean Utke, and Andrea Walther. Evaluating higher derivative tensors by forward propagation of univariate taylor series. *Mathematics of Computation*, 69(231):1117–1130, 2000.

[34] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[35] Alfred Haar. Die minkowskische geometrie und die annäherung an stetige funktionen. *Mathematische Annalen*, 78(1):294–311, 1917.

[36] Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405:947 EP –, 06 2000.

[37] Rob Hall, Stephen E Fienberg, and Yuval Nardi. Secure multiple linear regression based on homomorphic encryption. *Journal of Official Statistics*, 27(4):669, 2011.

[38] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

[39] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[40] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[41] Konrad Knopp. *Elements of the Theory of Functions*, volume 1. Courier Corporation, 1952.

[42] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[43] Joseph Louis Lagrange. *Théorie des fonctions analytiques: contenant les principes du calcul différentiel, dégagés de toute considération d'infiniment petits, d'évanouissans, de limites et de fluxions, et réduits à l'analyse algébrique des quantités finies*. Ve. Courcier, 1813.

[44] M. M. Lau and K. H. Lim. Investigation of activation functions in deep belief network. In *2017 2nd International Conference on Control and Robotics Engineering (ICCRE)*, pages 201–206, 2017.

[45] Yann Le Cun, Ido Kanter, and Sara A Solla. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, 66(18):2396, 1991.

[46] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[47] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[48] Adrien Marie Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes: avec un supplément contenant divers perfectionnemens de ces méthodes et leur application aux deux comètes de 1805*. Courcier, 1806.

[49] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[50] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.

[51] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pages 855–863, 2014.

[52] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices

and learning with errors over rings. In *Advances in Cryptology – EUROCRYPT*, pages 1–23. Springer, 2010.

[53] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):43, 2013.

[54] Hartmut Maennel, Olivier Bousquet, and Sylvain Gelly. Gradient descent quantizes relu network features. https://arxiv.org/abs/1803.08367, March 2018.

[55] Paulo Martins, Leonel Sousa, and Artur Mariano. A survey on fully homomorphic encryption: An engineering perspective. *ACM Comput. Surv.*, 50(6):83:1–83:33, December 2017.

[56] Lee Mathews. Equifax Data Breach Impacts 143 Million Americans. `https://www.forbes.com/sites/leemathews/2017/09/07/equifax-data-breach-impacts-143-million-americans/#4cd26117356f`. [Online; accessed 21-Sept-2019].

[57] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5-6):555–559, 2003.

[58] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international*

*conference on machine learning (ICML-10)*, pages 807–814, 2010.

[59] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348. IEEE, 2013.

[60] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT*, pages 223–238. Springer, 1999.

[61] Giambattista Parascandolo, Heikki Huttunen, and Tuomas Virtanen. Taming the waves: sine as activation function in deep neural networks. 2016.

[62] Harry Pratt, Frans Coenen, Deborah M Broadbent, Simon P Harding, and Yalin Zheng. Convolutional neural networks for diabetic retinopathy. *Procedia Computer Science*, 90:200–205, 2016.

[63] Y. Qin, X. Wang, and J. Zou. The optimized deep belief networks with improved logistic sigmoid units and their application in fault diagnosis for planetary gearboxes of wind turbines. *IEEE Transactions on Industrial Electronics*, 66(5):3814–3824, 2019.

[64] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. 2018.

[65] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[66] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[67] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[68] Nicol N Schraudolph. On centering neural network weight updates. *Orr, GB and M uller, K.-R., editors, Tricks of the Trade. Springer Verlag, Berlin. To appear in Lecture Notes in Computer Science*, 1998.

[69] Microsoft SEAL (release 3.5). `https://github.com/Microsoft/SEAL`, April 2020. Microsoft Research, Redmond, WA.

[70] Wei Shen, Mu Zhou, Feng Yang, Caiyun Yang, and Jie Tian. Multi-scale convolutional neural networks for lung nodule classification. In *International Conference on Information Processing in Medical Imaging*, pages 588–599. Springer, 2015.

[71] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[72] Marshall H Stone. The generalized weierstrass approximation theorem. *Mathematics Magazine*, 21(5):237–254, 1948.

[73] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[74] Vinod Vaikuntanathan. Computing blindfolded: New developments in fully homomorphic encryption. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 5–16, 2011.

[75] Xin Wang, Yi Qin, Yi Wang, Sheng Xiang, and Haizhou Chen. Reltanh: An activation function with vanishing gradient resistance for sae-based dnns and its application to rotating machinery fault diagnosis. *Neurocomputing*, 363:88 – 98, 2019.

[76] Karl Weierstrass. Über die analytische darstellbarkeit sogenannter willkürlicher functionen einer reellen veränderlichen. *Sitzungsberichte der Köniiglich Preußischen Akademie der Wissenschaften zu Berlin*, 2:633–639, 1885.

[77] Wei Wu, Jian Liu, Huimei Wang, Fengyi Tang, and Ming Xian. Ppolynets: Achieving high prediction accuracy and efficiency with parametric polynomial activations. *IEEE Access*, 6:72814–72823, 2018.

[78] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.